Communication’s Layer

Software Requirements Specifications (SRS)

**Change Control Information**

|  |  |  |  |
| --- | --- | --- | --- |
| Version No. | Date | Changed by | Scope of Change |
| 1.0 | 4/08/10 | Meital | Initial revision |
| 1.1 | 2/09/10 | Yael | After reviewing previous version with Yehuda |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. INTRODUCTION 3](#_Toc277512081)

[*1.1* Context 3](#_Toc277512082)

[*1.2* Purpose 3](#_Toc277512083)

[*1.3* Product Overview 3](#_Toc277512084)

[1.4 Definitions And Abbreviations 4](#_Toc277512085)

[2. OVERALL DESCRIPTION 4](#_Toc277512086)

[2.1 System environment 4](#_Toc277512087)

[2.2 Product Functions 4](#_Toc277512088)

[2.3 User Characteristics 5](#_Toc277512089)

[2.4 Assumptions and Dependencies 5](#_Toc277512090)

[2.5 Multithreading 6](#_Toc277512091)

[3 6](#_Toc277512092)

[3.1 Functional Requirements Definition 6](#_Toc277512093)

[3.1.1 Overview 6](#_Toc277512094)

[3.1.2 Establishing connections between parties 6](#_Toc277512095)

[3.1.3 Securing the connections 8](#_Toc277512096)

[3.1.3.1 Key exchange only: Security in the presence of eavesdropping adversaries 9](#_Toc277512097)

[3.1.3.2 Authenticated key exchange (this is typically what we will want): 9](#_Toc277512098)

[3.1.4 Success function 10](#_Toc277512099)

[3.1.5 Communication functions 10](#_Toc277512100)

[3.1.6 Status of connections 10](#_Toc277512101)

[3.1.7 Dynamic parties 11](#_Toc277512102)

[3.2 External Interface Requirements 11](#_Toc277512103)

[3.2.1 User Interfaces 11](#_Toc277512104)

[3.2.2 Communications Protocols 11](#_Toc277512105)

[3.2.3 Memory Constraints 11](#_Toc277512106)

[3.3 Software System Attributes 12](#_Toc277512107)

[3.3.1 Reliability 12](#_Toc277512108)

[3.3.2 Availability 12](#_Toc277512109)

[3.3.3 Security 12](#_Toc277512110)

[3.3.4 Maintainability 12](#_Toc277512111)

[3.3.5 Portability 12](#_Toc277512112)

[3.3.6 Performance 12](#_Toc277512113)

[3.4 Database Requirements 12](#_Toc277512114)

[3.5 Other Requirements 12](#_Toc277512115)

[4 ADDITIONAL MATERIALS 12](#_Toc277512116)

## INTRODUCTION

* 1. Context

The communication layer sits between the non-interactive and interactive mid-crypto layers of the Crypto SDK. It uses cryptographic tools such as digital signatures and encryption in order to provide secure and private channels. On the other hand, this layer is heavily used by the interactive mid-crypto protocols and MPC protocols.

* 1. Purpose

This layer needs to provide communication services to **any** interactive cryptographic protocol.

The implementation of this layer is hidden (black box) from the users. Users only know that they can request send and receive services to any of the parties involved in the computation/protocol. These services have different levels of security and privacy that are set by the requesting client (protocol).

Moreover, this layer is responsible for the establishment and the maintenance of the channels between the different pairs of parties involved. We refer later on in this document to whether we require the connections to be a clique or not.

* 1. Product Overview

In order to achieve the goals explained above, we identify that there exist three logical sub layers:

* + 1. The actual communication sub layer.

It sets the communication protocols such as TCP/IP via sockets or other implementation of communication.

* + 1. The parties’ manager.

This sub layer is responsible of building and managing the channels between all the parties. Meaning, support send and receive request to and from a specific party.

* + 1. Cryptographic communication.

It provides privacy and authenticity channels using the Crypto SDK.

From now on whenever we mention “layer” we refer to the overall communication’s layer, which includes these three sub-layers.

* 1. Definitions And Abbreviations

|  |  |
| --- | --- |
| Crypto SDK | Cryptography Software development kit |
| MPC | Multi party computation |
| Cryptographic material | 1)      A joint human-memorisable password (for ex: from a dictionary of size 1,000,000 words)  2)      A joint symmetric key (for ex: a 128-bit random string)  3)      Public keys: each party has a public/private key pair and knows the public-key of the other party |

## OVERALL DESCRIPTION

### System environment

This layer is part of the Crypto SDK. Therefore, it should be platform independent as the SDK. Thus, the Operating Systems can be any version of Windows, Linux, Unix or Mac which supports TCP/IP protocols. The development and testing will be performed in Windows XP or 7. We will not use libraries that do not support other OSs.

Even though this layer is part of the Crypto SDK, it only needs to use the lowest layer in it (the non-interactive primitives). Therefore, we require that anyone could instantiate and use this layer, regardless of the high-level layers.

### Product Functions

This product is an API infrastructure for interactive protocols. Any protocol interested in sending and receiving messages in a private and secure environment can use this layer for that purpose.

It should provide the following functionalities:

* Establish physical connections between parties (e.g. TCP/IP).
* Securing the connections.
* Supply a way of quantifying/qualifying success in connecting all the parties. For example, whether we managed to create a clique or not, what percentage of parties did connect, etc.
* Send and receive messages between parties:
  + Plain send/receive
  + Authenticated send/receive
  + Encrypted send/receive
  + Authenticated and encrypted send/receive
* Give feedback on the status of the connections.

### User Characteristics

The users of this communication infrastructure are applications/protocols that send/receive messages. For example, the third layer of the SDK as well as the higher-level protocol – MPC.

### Assumptions and Dependencies

* Each party receives as input the list of the other parties participating in the computation as well as cryptographic material about them.
  + As a first step for establishing a connection, a party listening to incoming requests for connection might accept all requests.
  + As a second step, there might be an ID verification based on the cryptographic material of the connected party.
* Each party has an ID. The ID has to be unique. Typically, this ID could be the IP Address but it is not a requirement. Relate to the problem of some participants being behind a NAT device
* We assume that every party should be known by the same name ID to all the other parties for as long as the protocol runs.
* We assume that the list of parties participating in the computation is the same for all the parties. This list is given to all the parties as input.
* When running a protocol that is composed of many sub-protocols, we may need to consider a specific party differently in each sub-protocol. Thus, we will assign an alias to the parties. For example, the fifth party (out of n) in the main protocol may need to be the first one in one of the sub-protocols. In this case we may refer to it with an alias so that we can keep track of it easily.
* In MPC is not necessary that all the parties should be equal. There can be different roles. For example, in VSS one of the parties is the dealer.
* The role is part of the information of the party, together with its IP Address and credentials. The role and the alias mentioned above are the same.
* Sometimes the order of the participants can be of importance. For example, we might refer to the first one as a special one.
* The communication layer is in charge of trying to establish connections between the parties received as input for a specific computation as well as providing the means to qualify the success of this activity. That is, it should provide different implementations of a “Success” function, which will give the level of success in establishing the connections. However, it is the responsibility of the protocol to decide what level of success is necessary. Thus, the application will use the necessary “Success” function.

The “Success” function is defined as follows:

**Input**: my established connections

**Output**: a subset of the established connections that we need to talk to.

* How long to try to connect to all the parties should be a parameter.
* If in the middle of an MPC a connection closes, it is the responsibility of this layer to notify the application running the protocol. The protocol will decide if to continue or not.

### Multithreading

The communication layer uses multi-threading very heavily. In fact, the connection to each other party should be performed in a different thread. Therefore, special attention has to be given to synchronization and interaction between threads.

3 SPECIFIC REQUIREMENTS

### Functional Requirements Definition

### Overview

The communication layer provides three main functionalities described above as three different logical sub-layers. These functionalities are:

* Physically connecting between parties
* Managing these connections
* Providing privacy and security throughout the communication

To dispel any doubt, the communication layer is used (instantiated) in each one of the parties that wish to participate in the computation. In fact, an application running (possibly) in a different computer represents each party. The application becomes a synonym of a party. Each application uses the communication layer in order to establish connections to all other parties. Each party performs this process independently of the others. How to synchronize between them is in the scope of the SDD and not the SRS.

At the end of the stage of establishing and securing the connections, each party knows with a 100% of certainty only what it itself managed to do. That is, it knows which parties it is connected to. Since an instance of the communication layer exists for each application, the communication layer does not have a general view of what is happening. It rather has a local view.

### Establishing connections between parties

We define “establish a connection” as either to accept or initiate a connection between my party and another.

INPUT: list of participants

Each party gets a list of parties participating in this computation as input. The information includes the IP Address and any necessary credentials, such as public keys (or other), of all the participants.

For each party that my party needs to connect to there will be a separate thread trying to do this. It is in the scope of the SDD to decide which threads will be listening for connections and which will be trying to connect.

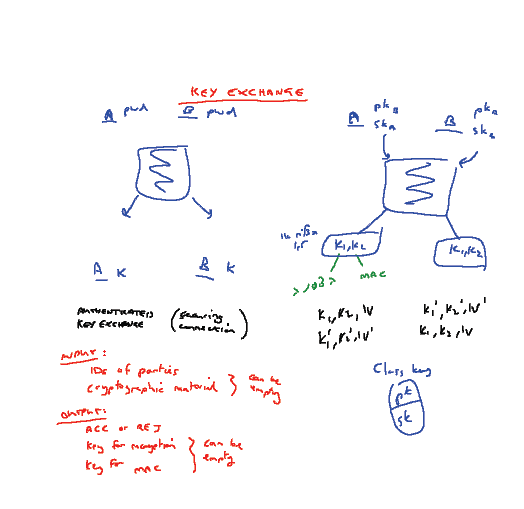
OUTPUT: list of established connections

This function tries for a certain period of time (time-out parameter) to establish physical connections to other parties in the list.

If all connections were established or the time-out has arrived this function returns a list of the established connections.

This list is a shared resource between all the threads. It needs to be synchronized since each thread will update it as soon as it manages to establish the connection.

### Securing the connections

At the connection’s stage there is no way of knowing whether the party that is trying to connect to us is a valid one or not. We need to verify identities of parties connecting, according to their credentials. Only after the connection is set, the parties will be able to **exchange keys** and perform **verification of** their **identities**.

### Key exchange only: Security in the presence of eavesdropping adversaries

INPUT: nothing

Output: a pair of symmetric keys (one for encryption and one for MAC)

Both parties have the same keys so that they can encrypt/decrypt and MAC/verify the messages sent afterwards.

Here there is no identity authentication

### 3.1.3.2 Authenticated key exchange (this is typically what we will want):

INPUT: cryptographic material (joint secret password, joint symmetric key, public keys for both participants)

OUTPUT: a pair of symmetric keys (one for encryption and one for MAC) and a bit saying ACCEPT or REJECT.

Both parties have the same keys so that they can encrypt/decrypt and MAC/verify the messages sent afterwards.

The ACCEPT/REJECT bit signifies whether I am convinced that I am talking to the "right person".

The three different possibilities of cryptographic material (see [section 1.4](#Definition)). give three different types of key exchange, based on three different types of inputs.   
Based on what the cryptographic material is, the "right person" differs. That is, in cases (1) and (2), the "right person" is someone who holds the same password/key as me. In case (3), the "right person" is someone who knows the secret key associated with the public-key of the other party that I am holding. The key exchange protocol ensures that this property is upheld (i.e., a party who does not have this information cannot get me to ACCEPT).

### Success function

Ideally, each two parties in the list should connect, and at the end of the connecting stage, we should obtain a clique.

However, for different reasons not always we will have a clique. It could be because a party or more are malicious, or just simply a technical problem.

It is up to each cryptographic protocol to decide whether it can run or not without a clique and under which conditions.

The communication layer will provide different implementations of a “Success” function. The application running the protocol will use the appropriate one according to the protocol’s specification. To illustrate this, Protocol A may require a clique whereas Protocol B may be able to work with 90% of connections. Furthermore, there is an issue of credibility. One of the implementations of the “Success” function will have to deal with this issue. The problem is how we know if some party is lying about the connections it has managed to establish. For some protocols, a naïve approach may be enough. That is, each party sends the list of connections and we believe him. Nevertheless, for other protocols a secure broadcast of this list may be required.

### Communication functions

This layer should provide its client an API to send and receive messages to any other party.

We identify four forms of sending and four forms or receiving messages

* Plain
* Encrypted
* Authenticated
* Encrypted and authenticated.

It might be the case that a protocol requires the communication to be encrypted, but some of the messages sent during the protocol do not need to. In that case, for the sake of optimization we require to “turn off” the security ability.

There are many encryption and authentication algorithms. The programmer will choose at the time of building the application the desired ones.

### Status of connections

This function returns the updated list of connections. This function can be called at any given time.

### Dynamic parties

For the majority of protocols it is safe to assume that the list of parties participating is static. If one or more parties disconnect during the course of the protocol, is up to the protocol to decide whether to continue without the disconnected parties or to abort.

However, there are protocols that need to allow new parties to connect while running. Therefore, we need to allow parties to connect dynamically.

The dynamic addition of parties is not done in a chaotic way. We can assume two restrictions to it:

* All the parties know in advance the list of overall parties participating. (Those that connect in the very first stage, and those that might connect in later stages).
* Parties cannot connect at any given time. Instead, there will be “connecting stops” indicated by the running protocol. That is, as part of the protocol it might request to allow now the addition of new parties.

This implies that we should be able to call the establish connection functionality followed by success checking at the time requested by the protocol.

### External Interface Requirements

### User Interfaces

No GUI interfaces needed for this layer.

### Communications Protocols

* + - 1. On a physical level, the communication layer will use TCP/IP protocols.
      2. On a conceptual level, the messages sent from one party to another are synchronized. That is, usually one party will wait to get a message from the other party before sending its own. However, it is the responsibility of the calling protocol to synchronize these messages and not of the communication layer.

### Memory Constraints

Throughout the whole running of the Protocol, this layer will keep a record of all the connections. This will include information about the IP address, role, etc. This information will be shared by all the threads in charge of the actual connections. It does not seem that this will be a huge memory constraint.

### Software System Attributes

### Reliability

As an infrastructure, this layer must be reliable and report to the user on any connectivity error.

### Availability

This layer is part of the Crypto SDK. It only uses the crypto non-interactive layer of the SDK. It can be used independently of the higher levels of the SDK. Therefore, there is no need to install the whole SDK in order to use it. (Suggestion: if we want to offer a platform of communication that doesn’t necessarily need to be secure but knows how to manage connections between various parties we could “Switch off” the crypto capabilities. TO DO: decide if we want this)

### Security

This comm. Layer provides security of the channels according to the protocols demands.

On a different level, it also should be securely coded to prevent secure coding attacks. This should also be considered when deciding how to send messages. Messages should be sent as objects and not as serialized data. Moreover, the message object should include only absolutely necessary information.

### Maintainability

### Portability

### Performance

### Database Requirements

### Other Requirements

## 4 ADDITIONAL MATERIALS